ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

Implementing a Performance Improved Controller for SoC

[1] Vellampati Harathi

***Abstract*— With increasing various complexities of semiconductor devices due to growing performance, functionality requirements and with diminished time to market, the semiconductor firms try to develop null defect products in very less development time [1]. In SoC control unit plays a vital role and it is responsible for data transfers between blocks of the system, initialization and configuration, programming, power management etc. The processor present in the control unit executes the firmware from non-volatile memory (ROM). Replacement of the firmware might be required if there are defects in the pre-loaded code or if the additional feature is need to be implemented but replacing the firmware is very tedious and time consuming task and also it requires additional fabrication steps which could prove costly. Due this reason for the incorrect functions present in the firmware can be corrected with the expected functionality in Private Non Volatile Memory (PNVM) as a separate patch. This paper discusses the PNVM patch implementation.**

Index terms— less time to market, Low cost, G5 SoC, G5 Controller, patch creation.

I. INTRODUCTION

Integrated circuits have undergone dramatic changes with increasing the complexity of electronic devices. Due to increase in the complexity and cost, design engineers came up with novel design methodologies. System on chip technology is one of the new technologies. The difficulties present in the other technologies like system on boards and system in package can be eliminated in the system on chip technology and the system on chip technology gives increase system complexity along with good flexibility [2]. The present semiconductor industry is adopting the system on chip integrated circuits to implement highly complicated systems due to its small size, less cost and less power consumption.

The key to victorious adoption of system on chip technology as a global standard is the reduction of development cycle time of the system. As already mentioned above the main reason for the reduction of system development time is due to the reusability of pre designed and pre verified intellectual property blocks and these blocks will be combined on the single semiconductor [3]. In any electronic system development process most of the time will be spent for verification process only, hence the time required for the verification process will be reduced due to the usage of pre verified intellectual property blocks.

In any system controller place a crucial role and the efficiency of the system can be determined based on the controller's performance. A controller is nothing but a processor (example microprocessor) built with electronic techniques. Depending on the number of inputs and outputs the system may contain more than one control unit. The controller plays major role in the operation of the system. The control unit performs device initialization and configuration, power management, providing proper clocking support to the reset of the parts of the system. The controller is responsible for the data movements among the different components of the system. The processor executes the firmware from the read only memory. Once the firmware is designed, it

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

may be rarely replaced because changing the firmware of device needs the complete device replacement.

In the previous system on chip designs (G4), if any bugs were present in firmware or to add any additional features, the designers used to replace the firmware of the design. Hence replacing the firmware of the design needs extra fabrication steps. To avoid all the pitfalls in the present system on chip designs (G5) a separate patch will be provided for incorrect functionalities and additional features without replacing the actual firmware of the design. The firmware execution will start from ROM (read only memory) and jump to private non-volatile memory (PNVM) where patch will be present and again execution will be returned to original flow. This minimizes the system development cycle time and cost.

This paper contains the fundamentals of G5 System on chip, PNVM patch, PVNM memory read and writes, patch creation and patch verification.

II. FUNDAMENTALS OF G5 SoC

The basic details of G5 system on chip are narrated by outlining the functionality of all the sub modules present in it. G5CONTROL and PNVM modules are explained broadly because these are the major blocks in the current project.

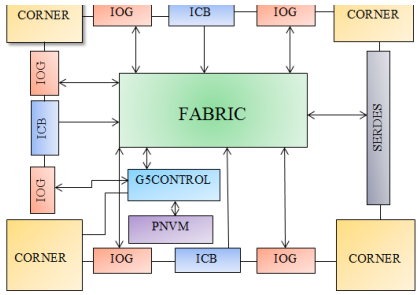

Fig [1]: The Block Diagram of G5 SoC

Fabric is the core of the G5 Full chip device, where the end user can implement their design. In the G5 SoC the fabric block is a field programmable gate array (FPGA) and remaining blocks are application specific integrated circuits (ASIC) [4]. The SoC's are replacement to the bulky and high power consumed systems built by discrete components.

FPGA Fabric

The FPGA fabric is a non-volatile (Sonos-based) custo block, which facilitates the implementation of programmable user logic. The major blocks of the fabric are: 1.LSRAM (Large SRAM), 2.USRAM (micro SRAM), 3.UPROM (Micro PROM), 4.MATH 5.PG (Pulse Generator).

SERDES

There may be multiple instances of the SERDES PHYon a G5 device, some of which are standalone (GPSS) and connect straight to the fabric, whilst others are tightly coupled to a dual PCIe controller block (PCIESS).

G5CONTROL

G5CONTROL is the control block of G5 and implements the following functionality:

- Initialization and configuration
- Programming
- Power management
- User debug
- Security

- *Initialization and configuration*: Device initialization (reset and configuration) is

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

- required at both at power-up (cold reset) and warm reset events. At power-up, there are three of stages of device initialization, which are: 1. Power on Reset, 2. Boot, 3.UIC Script Execution.

- ***Programming:*** In G5 SoC two programming modes are available, which are: 1. JTAG Programming, 2.SPI Slave Programming.
  JTAG Programming: JTAG programming mode uses the JTAG interface to receive JTAG programming instructions from an external JTAG master.
  SPI Slave Programming: SPI Slave programming mode uses the G5CONTROL SPI in slave mode. Programming commands are received from an external SPI master.
- *Power management*: G5 contains functionality to support both design-time and run-time management of both static and dynamic power in the user's system.
- User debug: It is required that in some cases the user have the ability to interactively peek and poke the internal values in an uSRAM or an LSRAM whilst the user's system continues to operate. But the debug accesses require access to the SRAM via the FCB, which is not allowable at the same time as the user is accessing the SRAM. A handshake is required between the user and the FCB in order to allow the two to share an SRAM interactively.
- *Security*: The System Controller has direct access to all onboard storage elements, which
- offers a powerful system for analyzing and manipulating the state of the FPGA for system debug. However, this is of equal value to an attacker and thus must be protected. The System Controller has direct access to all onboard storage elements, which offers a powerful system for analyzing and manipulating the state of the FPGA for system debug. However, this is of equal value to an attacker and thus must be protected.

IO Clock Bridge (ICB)

Every side of G5 device contains either one or two IO Clock Bridges, which allow multiplexing of clocks from various sources and provide an entry point into the clock resources of the FPGA fabric [4]. The east and west side always have one ICB, although the ICB on the east side may be slightly different to the others, as it is associated with SERDES clocks.

IO Gearing (IOG)

Each of the General Purpose IO (GPIO) and High Speed IO (HSIO) banks in the device has associated IO gearing functionality. This logic handles gearing of multiple FPGA fabric data bits to/from a single device IO, to accompany a division of clock frequency between the IO and the fabric. It also handles potential sharing of each IO cell with a dedicated logic function other than the FPGA fabric and the boundary scan functionality of the IO.

Corner

The base corner block contains two PLLs and two DLLs, two configurable delay lines, the associated clock routing multiplexors to route signals to and from the PLLs and DLLs, and the IO bank power-on detect circuits used to control the adjacent IO banks and one

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

clock calibration block used to test the PLLs and DLLs in each corner. Most of the control signals for the PLLs, DLLs, Delay Lines and Multiplexors come from the integrated corner register map. All others come from the EIP interface from FPGA routing signals.

III. PVNM PATCH

PNVM

Connected to the G5C Controller is 1Mbit (128K Byte) PNVM (Private Non Volatile Memory) that allows the device firmware to be updated without a new mask set being created. This is only directly accessible by G5CONTROL, but sections of it may be used indirectly by the user for storage of user data, such as keys.

To support these multiple uses the PNVM is split into four independent segments that can be erased and programmed independently with no chance off disturbance.

Table [1]: PNVM Sectors

Sector	Size
0	56 K Bytes
1	56 K Bytes
2	8 K Bytes
3	8 K Bytes

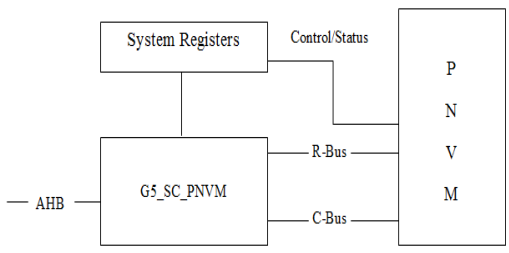

Fig [2]: PNVM Controller

The PNVM controller in G5C implements an AHB interface to the PNVM R and C interfaces [5]. The C-Bus (32-bit) is used for programming operations and the R-Bus (64-bit) for read operations. The system registers allow the sleep and power controls of the NVM to be controlled, and also configure the PNVM controller.

R BUS Operation

Normal read operation includes setting up valid addresses and issuing the read clock. The R_bus is used to perform read operation on the flash memory. It uses the following signals to communicate a valid read operation: 1.r_valid, 2.r_grant, 3.r_q[63:0], 4.r_addr[14:0].

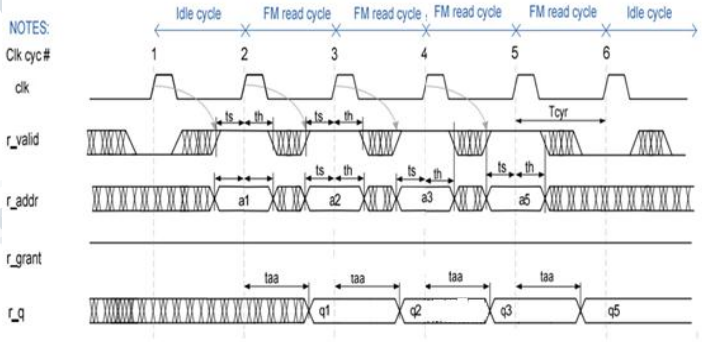

Fig [2]: R_Bus Read Cycles

C BUS Operation

The C bus is used to read and write to the entire register map as well as write to the page latches. These registers are addressed using a 9 bit address signal called c_addr[8:0]. The signals used by the c_bus to perform any operation are: 1.c_size[1:0],2.c_addr[8:0],3.c_d[31:0],4.c_valid,5.c_gra

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

nt,6.c_grant,6.c_write,7.c_q[31:0],8.c_q_valid,9.c_clk_req,10.c_irq.

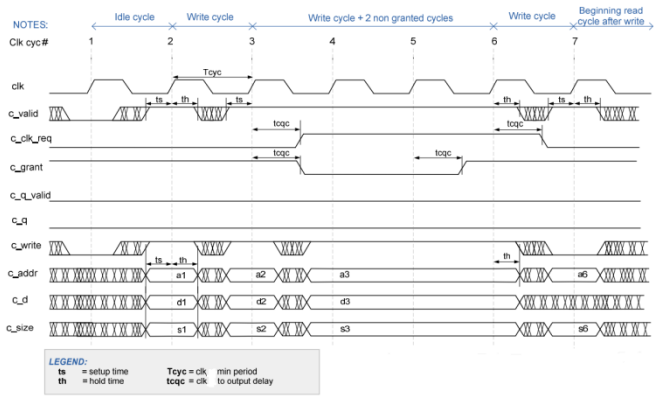

Fig [3]: C_Bus Read Cycles waveform

The write operation also performed in the same way.

PNVM Operations

The pNVM may be used to install a patch at boot of a non-virgin device. For maximum flexibility the patch acts as its own boot-loader allowing it to execute directly from pNVM or it may copy itself to RAM for faster execution. One of the likely scenarios for requiring a patch is to modify the device boot sequence. However, a patch may not be installed until the pNVM has been enabled. Thus a patch cannot be used to modify behavior of earlier parts of the boot sequence. To indicate the presence of a patch, a 32-bit marker is placed in the pNVM. If this marker is present then the patch is assumed to be present and control is transferred to the patch.

- PNVM Initialization: The initialization routine will read out the parameters necessary for programming the PNVM and store them into SRAM. It uses parameters necessary to read the PNVM which are stored in the UFS user factory segment.
- PNVM Load Page Latch: This operation loads data to be programmed from SRAM into the PNVM block programming latches.
- PNVM Program Page: This is the top level program operation of the PNVM. It will write the data that was previously loaded into the page latches into the selected sector and page. It will perform a pre-program, erase and program cycle to write to the PNVM. After the programming cycle is complete a verify operation is performed either at 0 or at beginning of life limits.
- PNVM Sector Erase: This operation applies an erase pulse to the selected sector. This operation is used during zeroization only.
- PNVM Sub Sector Erase: This operation applies an erase pulse to the selected sub sector. The PNVM can select 8 pages at a time to erase. This is what is referred to as a sub sector. This operation is used during zeroization only.
- PNVM Sector Program: This operation applies a program pulse to the selected sector. This operation is used during zeroization only.
- PNVM Program Row: This operation applies a program pulse to the selected sector and selected page. It is performed only on one page.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

IV. PATCH CREATION AND VERIFICATION

Step 1: Basic Cortex M3 boot from ROM.

Booting of Cortex M3 can be checked by generating a TXEV pulse using assembly code and check if that pulse is getting triggered.

```
fork
  begin
    @(posedge `DUT_TOP.g5c.UATPG.UWBR.UMAIN.txev);
     t1 = $realtime();
    @(negedge `DUT_TOP.g5c.UATPG.UWBR.UMAIN.txev);
    t2 = $realtime();
    #100ns;
    if(t2-t1 > 50ns)
      begin
        `uvm_info("TESTCASE", "TXEV check passed", UVM_NONE);
      end
    else
      begin
        `uvm_error("TESTCASE", "TXEV check failed");
      end
  end
  begin
    #600us;
    `uvm_fatal("TESTCASE", "TXEV check timeout");
  end
join_any
disable fork;
`uvm_info("TESTCASE", "Testcase completed", UVM_NONE);
```

Fig [4]: Code to verify Cortex M3 boot

Step 2: Access PNVM control space and check all the registers in it. The PNVM has 19 control registers to store the data required for configuration. All the registers are 32 bit in size. The control registers can be read and written through c Bus.

Power on Reset (POR) values of PNVM control registers:

```
PRINT_MSGF(LOG_INFO,"Read POR Values of PNVM Registers");
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x00);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x04);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x08);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x0C);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x10);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x14);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x18);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x1C);
```

Fig [5]: Reading POR values of control registers

TIMER_CFG_ADDR: This register is used to configure and start the timer. Its POR value is h38000000.

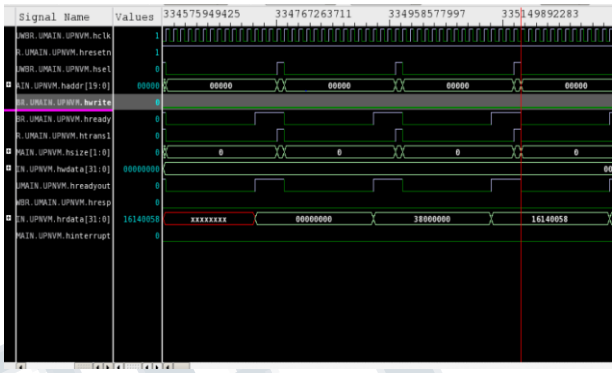

Fig [6]: Power on Reset values of control registers

Registers contents written with user defined values.

```
PRINT_MSGF(LOG_INFO,"Write/Read Values of PNVM Registers");
PRINT_MSGF(LOG_INFO,"Write/Read for PNVM REG-1");
HW_set_uint32(PNVM_CONTROL_SPACE_ADDR,0x00,0xFFFFFFFF);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x00);
PRINT_MSGF(LOG_INFO,"Write/Read for PNVM REG-2");
HW_set_uint32(PNVM_CONTROL_SPACE_ADDR,0x04,0xFFFFFFFF);
read_data = HW_get_uint32(PNVM_CONTROL_SPACE_ADDR,0x04);
```

Fig [7]: Write/Read values of control registers

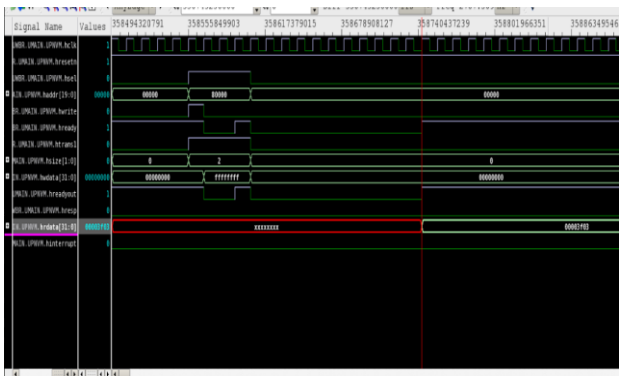

Fig [8]: Written values of control registers

Step 3: PNVM memory access.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

The memory can be written through c bus and read can be done through R bus. Here we are writing some random values to memory. The memory values can be observed on AHB signals.

```
        for (byte_addr=0; byte_addr<4; byte_addr++) begin
          ahb_trans(AHB_WRITE, write_transfer, 32'h38080000, AHB_SINGLE, AHB_BITS_32, {18'h0,pa[5:0],5'h0,byte
          ahb_trans(AHB_WRITE, write_transfer, 32'h38080004, AHB_SINGLE, AHB_BITS_8, $urandom());
end


for (j=0; j<64; j++) begin
                ahb_trans(AHB_READ, write_transfer, pnvm_register_address, AHB_SINGLE);
                pnvm_register_address = pnvm_register_address + 4;
end
```

Fig [9]: PNVM Memory read/write

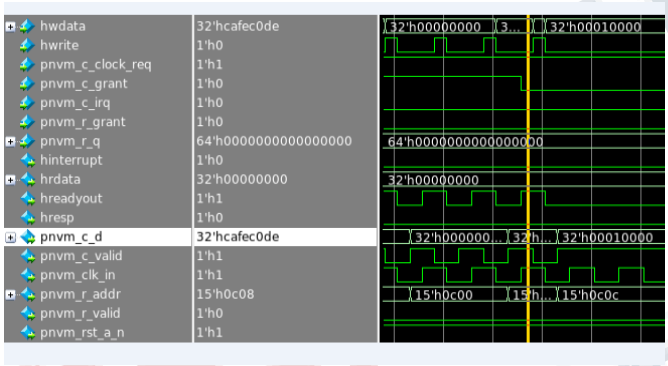

Fig [10]: PNVM Memory writes

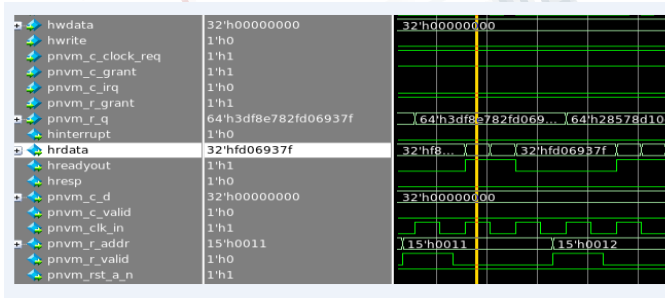

Fig [11]: PNVM Memory reads

Step 4: Patch creation.

Patch is nothing but a specific function written in C. It will be placed in the PNVM memory. Status indicator tells the patch presence. Using the IAR tool we will create .hex file of the patch and it will be placed in the specific location. The booting will be initiated in the ROM and it will jump to PNVM and executes particular function which is placed in the PNVM memory and comes back to original boot flow.

Patch installation:

Entry	Address	Size	Type	Object
svc_ff_entry	0x10004689	0xbc	Code Gb	patch_svc.o
sys_idle_mode	0x100030ed	0x54	Code Gb	patch_sys.o
sys_prog_mode {Abs}	0x0000b999	0x6c	Code Gb	g5rom.symbols

Fig [12]: Functions and their address

```
void loader(void)
{
 uint32_t* dst = (uint32_t*) &vector_table;
 const uint32_t* src = patchdata;
 const size_t nWords = (3u + (size_t) &eofmarker - (size_t) dst) / 4u;
  wdog_kick();

#if 0
  patch_serial_init();
#endif
 PATCH_PRINTF("Copying patch from %p to %p size %u words\n", src, dst, nWords);

 copy_image(dst, src, nWords);
 PATCH_PUTS("Copy complete, installing");

 // Update the MPU to allow EXEC on SRAM code space

 MPU->RNR = 2u;    /* SRAM (code space) 64KB R/W */
 MPU->RASR &= ~MPU_RASR_XN_Msk;

 // Install the new FIT

 SCB->VTOR = (uint32_t) &vector_table;    // set up FIT
  __DSB();
 // Execute the patch startup function.
 if (FIT->sp != 0u) {
  __set_MSP(FIT->sp);          // reset stack pointer
  FIT->startup();
  while (1);                   // startup should not be allowed to return
 } else {
  FIT->startup();
 }
}
```

Fig [13]: Patch installation

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 6, June 2017

```
00000070   0x6829           LDR     R1,[R5, #+0]
00000072   0x2002           MOVS    R0,#+2
00000074   0xF360 0x210A     BFI     R1,R0,#+8,#+3
00000078   0x6029           STR     R1,[R5, #+0]
                              TRITON_SYSREG->IOCTL.DCE = 7u;
                              sys_prog_mode(true, false);
0000007A   0x2100           MOVS    R1,#+0
0000007C   0x6828           LDR     R0,[R5, #+0]
0000007E   0xF440 0x60E0    ORR     R0,R0,#0x700
00000082   0x6028           STR     R0,[R5, #+0]
00000084   0x2001           MOVS    R0,#+1
00000086   0xE001           B.N     ??svc_ff_entry_5
```

Fig [14]: Firmware execution without patch

```
00000076   0x6829           LDR     R1,[R5, #+0]
00000078   0x2002           MOVS    R0,#+2
0000007A   0xF360 0x210A     BFI     R1,R0,#+8,#+3
0000007E   0x6029           STR     R1,[R5, #+0]
                              TRITON_SYSREG->IOCTL.DCE = 7u;
00000080   0x6828           LDR     R0,[R5, #+0]
00000082   0xF440 0x60E0    ORR     R0,R0,#0x700
00000086   0x6028           STR     R0,[R5, #+0]
                              sys_idle_mode();
                              break;
```

Fig [15]: Firmware execution with patch

V. CONCLUSION

The incorporation of patch in the present system on chip devices is novel approach. It plays a crucial role in reducing the time to market of the device. In many of the system designs once the firmware is designed it can't be changed because replacing the firmware is tedious job. By providing the separate patch for the incorrect and additional functionalities of the firmware can eliminate the firmware redesign. Due to the elimination of the extra fabrication steps the cost of the device will be reduced and time to market of SoC will less.

VI. REFERENCES

[1] Rohit Srivastava, Nandini Mudgil, Gaurav Gupta "SoC Time to Market Improvement through Device Driver Reuse: An Industrial Experience", Electronic System Design (ISED), 2012 International Symposium ,year:2012.

[2]http://eecs.wsu.edu/~pande/Journal_Papers/Paper_IEEE_Proceedi ngs.pdf.

[3]http://www.semiconductorstore.com/blog/2015/System-on-Chip-vs-Single-Board-Computer-A-Comparison-Guide/689.

[4]http://www.microsemi.com/products/fpga SoC/fpga/igloo2-design-resources-archive.

[5]J. R. Cricchi; D. A. Barth; H. G. Oehler; R. C. Lyman; J.M. Shipley; B. Ahlport "Radiation Hardened CMNOS/SOS Mask Programmable ROM and General Processor Unit", IEEE Transactions on Nuclear Science, Year:1977,Volume:24, Issue:6 Pages:2236-2243.

Vellampati Harathi

Received her B.Tech degree in Electronics & Communication Engineering from Jawaharlal Nehru Technological University Pulivendula, kadapa Dist., A P., India in 2014 and is shortly finishing her M Tech degree in VLSI Design & Embedded Systems from R.V. College of Engineering (VTU, Belgaum), Bangalore, Karnataka, India. Her interest areas are VLSI Design, digital electronic.